


Tópicos Especiais em Engenharia de Software (Jogos II)

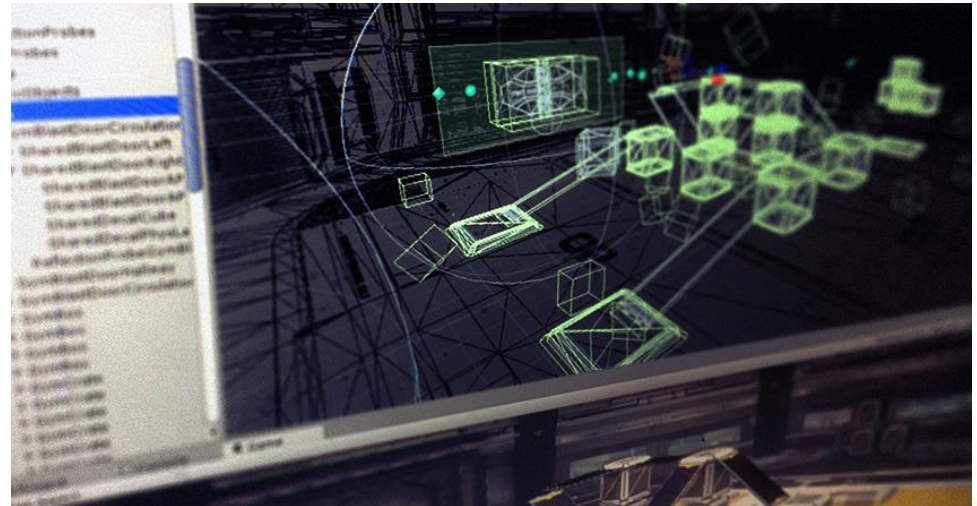
Aula 05 – Física

Edirlei Soares de Lima
<edirlei@iprj.uerj.br>



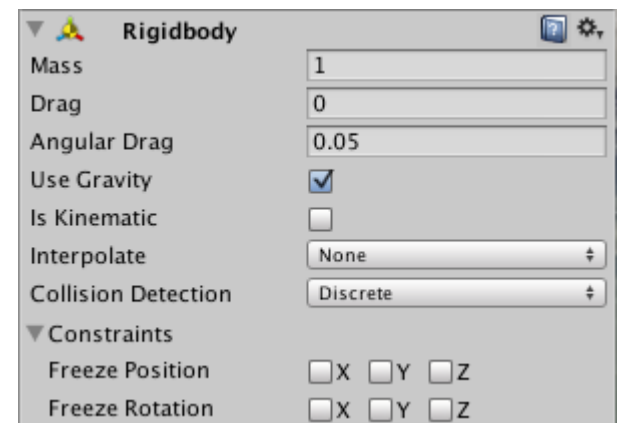
Unity 3D: Scripting

- A engine física da Unity fornece todos os componentes para simulação física realista.
 - Rigidbodies
 - Colliders
 - Triggers
 - Physics Forces e Torque
 - Physics Materials
 - Physics Joints



Rigidbody

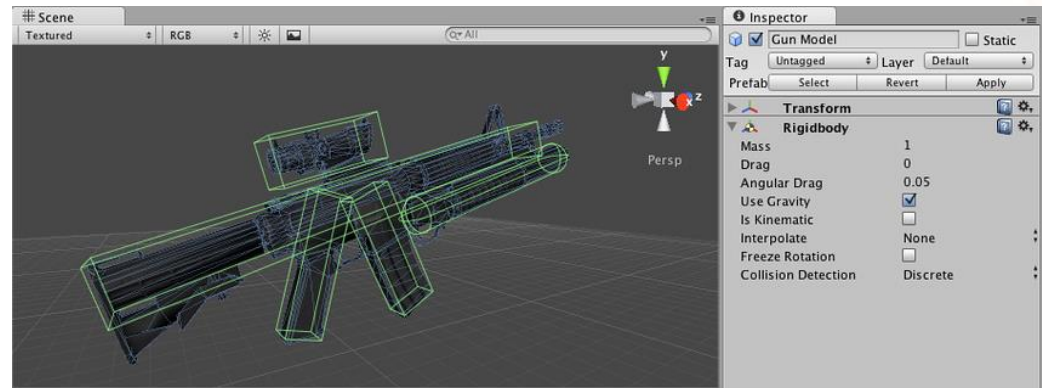
- Um **Rigidbody** é o principal componente que permite que um objeto possua um comportamento físico.
 - Com um Rigidbody, o objeto irá responder imediatamente a gravidade e forças físicas aplicadas sobre ele.
- **Adicionar um Rigidbody a um objeto:** Component -> Physics -> Rigidbody
 - **Mass:** massa do objeto em kg;
 - **Drag:** resistência do ar;
 - **Angular Drag:** resistência do ar;
 - **IsKinematic:** habilita/desabilita a atuação da engine física no objeto;



<https://docs.unity3d.com/Manual/class-Rigidbody.html>

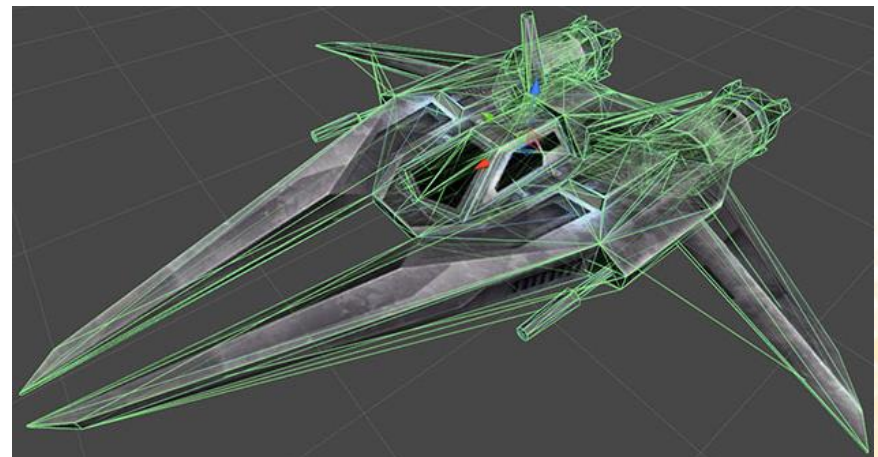
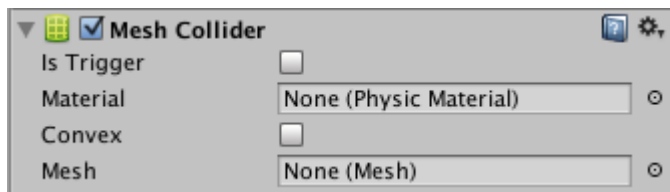
Colliders

- Os **Colliders** definem as formas dos objetos para colisões físicas. São invisíveis e não precisam ser exatamente iguais as malhas dos objetos (aproximações são mais eficiente e indistinguíveis durante o gameplay).
- **Adicionar um Collider a um objeto:**
Component -> Physics -> (*Type*) Collider
- **Colliders primitivos:**
 - Box Collider;
 - Sphere Collider;
 - Capsule Collider;



Colliders

- Em casos onde Colliders compostos não são suficientemente precisos para representar a forma correta dos objetos, é possível utilizar **Mesh Colliders** para representar a forma exata da malha do objeto.
 - Processo de verificação de colisão complexo.
 - Não existe colisão entre Mesh Colliders (solução: ativar opção Convex).

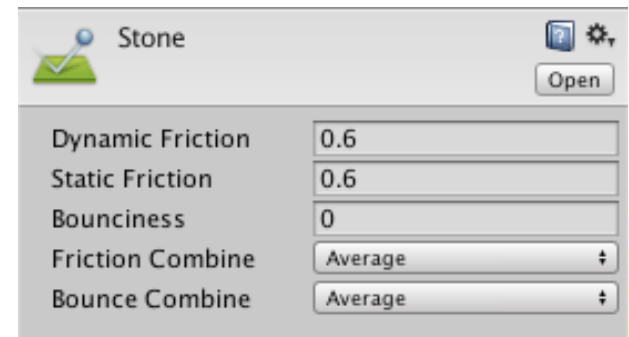


Colliders

- Colliders podem interagir com outros Colliders de diferentes maneiras dependendo da forma como os seus componentes são configurados.
 - **Static Collider:** Um GameObject sem Rigidbody e com Collider é estático. Geralmente usados para o cenário. É importante que esses objetos permaneçam completamente parados durante o gameplay.
 - **Rigidbody Collider:** Um GameObject com Rigidbody e com Collider é dinâmico e controlado pela engine física da Unity.
 - **Kinematic Rigidbody Collider:** Um GameObject com Collider e Rigidbody Kinematic (propriedade) interage corretamente com outros objetos físicos (pode ser movido durante o gameplay), mas não é afetado por forças físicas.

Physics Material

- Quando Colliders interagem, as suas superfícies precisam simular as propriedades do material que supostamente representam.
 - **Exemplos:** uma superfície de gelo é escorregadia, uma bola de borracha oferece mais atrito e um alto fator de elasticidade.
- **Criar um novo material:** Assets -> Create -> Physic Material
 - **Dynamic Friction:** fricção usada quando o objeto está em movimento;
 - **Static Friction:** fricção usada quando o objeto está parado;
 - **Bounciness:** fator de elasticidade;



Detectando Colisões por Scripts

- Quando ocorrem colisões, a Unity chama automaticamente funções de eventos colisão em todos scripts anexados aos objetos envolvidos.

```
void OnCollisionEnter(Collision collisionInfo)
```

```
void OnCollisionStay(Collision collisionInfo)
```

```
void OnCollisionExit(Collision collisionInfo)
```


Detectando Colisões por Scripts

```
public class PhysicsTest : MonoBehaviour {  
  
    void OnCollisionEnter(Collision collision){  
        if (collision.gameObject.tag == "Player"){  
            GetComponent<Renderer>().material.color = Color.green;  
        }  
    }  
  
    void OnCollisionStay(Collision collision){  
        if (collision.gameObject.tag == "Player"){  
            GetComponent<Renderer>().material.color = Color.red;  
        }  
    }  
  
    void OnCollisionExit(Collision collision){  
        if (collision.gameObject.tag == "Player"){  
            GetComponent<Renderer>().material.color = Color.blue;  
        }  
    }  
}
```

Triggers

- É possível detectar quando um objeto passa pela área física de outro objeto através de Triggers.
 - Propriedade “IsTrigger” nos Colliders.

```
void OnTriggerEnter(Collider collisionInfo)
```

```
void OnTriggerStay(Collider collisionInfo)
```

```
void OnTriggerExit(Collider collisionInfo)
```

Triggers

```
public class PhysicsTest : MonoBehaviour {

    void OnTriggerEnter(Collider collisionInfo){
        if (collisionInfo.gameObject.tag == "Player"){
            collisionInfo.gameObject.GetComponent<Renderer>().
                material.color = Color.green;
        }
    }

    void OnTriggerStay(Collider collisionInfo){
        if (collisionInfo.gameObject.tag == "Player"){
            collisionInfo.gameObject.GetComponent<Renderer>().
                material.color = Color.red;
        }
    }

    void OnTriggerExit(Collider collisionInfo){
        if (collisionInfo.gameObject.tag == "Player"){
            collisionInfo.gameObject.GetComponent<Renderer>().
                material.color = Color.blue;
        }
    }
}
```

Joints

- Joints são utilizados para conectar objetos fisicamente.
 - **Hinge Joint:** conecta dois objetos como se eles estivesse ligados por uma dobradiça. Ideal para representar portas, mas também pode ser usado para representar correntes, pêndulos, etc.
 - **Spring Joint:** simula a confecção de dois objetos através de um elástico.
 - **Fixed Joint:** conecta dois objetos de forma fixa (semelhante a hierarquia).
 - **Character Joint:** usado para criar Ragdolls.
 - **Configurable Joint:** incorpora as configurações de todos os outros joints, permitindo a criação de joints personalizados.

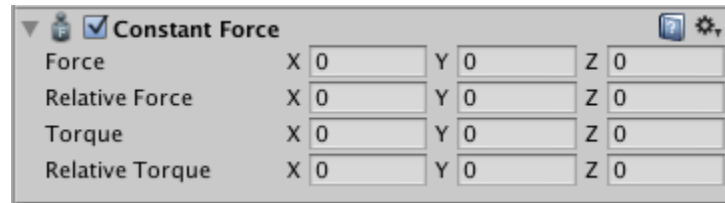
Character Controller

- O Character Controller é geralmente utilizado para controlar personagens.

```
public class ControlePersonagem : MonoBehaviour {
    public float speed = 70;
    private CharacterController ch;
    void Start() {
        ch = GetComponent<CharacterController>();
    }
    void Update() {
        if (Input.GetKey(KeyCode.UpArrow))
            ch.SimpleMove(transform.forward * speed * Time.deltaTime);
        if (Input.GetKey(KeyCode.DownArrow))
            ch.SimpleMove(-transform.forward * speed * Time.deltaTime);
        if (Input.GetKey(KeyCode.LeftArrow))
            transform.Rotate(Vector3.up * -speed * Time.deltaTime);
        if (Input.GetKey(KeyCode.RightArrow))
            transform.Rotate(Vector3.up * speed * Time.deltaTime);
    }
}
```

Forças

- É possível aplicar forças em objetos através de scripts ou através do componente “Constant Force”.



- Métodos do Rigidbody:
 - AddForce
 - AddForceAtPosition
 - AddTorque
 - AddExplosionForce
 - AddRelativeForce
 - AddRelativeTorque

Aplicando Forças

- AddForce:

```
public class PhysicsTest : MonoBehaviour {
    private Rigidbody rb;
    public float moveForce = 50;

    void Start() {
        rb = GetComponent<Rigidbody>();
    }

    void FixedUpdate() {
        if (Input.GetKey(KeyCode.UpArrow))
        {
            rb.AddForce(Vector3.forward * moveForce);
        }
    }
}
```

Aplicando Forças

- AddForceAtPosition:

```
public class PhysicsTest : MonoBehaviour {
    private Rigidbody rb;
    public float moveForce = 50;

    void Start() {
        rb = GetComponent<Rigidbody>();
    }

    void FixedUpdate() {
        if (Input.GetKey(KeyCode.UpArrow)) {
            rb.AddForceAtPosition(transform.forward * moveForce,
                                 new Vector3(transform.position.x,
                                             transform.position.y - 0.4f,
                                             transform.position.z));
        }
    }
}
```


Aplicando Forças

- AddTorque:

```
public class PhysicsTest : MonoBehaviour {
    private Rigidbody rb;
    public float moveForce = 50;

    void Start() {
        rb = GetComponent<Rigidbody>();
    }

    void FixedUpdate() {
        if (Input.GetKey(KeyCode.LeftArrow))
        {
            rb.AddTorque(transform.up * force);
        }
    }
}
```

Aplicando Forças

- AddExplosionForce :

```
public class PhysicsTest : MonoBehaviour {
    public float radius = 5.0f;
    public float power = 10.0f;

    void Update() {
        if (Input.GetKeyDown(KeyCode.Space)) {
            Vector3 explosionPos = transform.position;
            Collider[] colliders = Physics.OverlapSphere(explosionPos,
                                                         radius);

            foreach (Collider hit in colliders) {
                Rigidbody rb = hit.GetComponent<Rigidbody>();
                if (rb != null)
                    rb.AddExplosionForce(power, explosionPos, radius, 3.0f);
            }
        }
    }
}
```